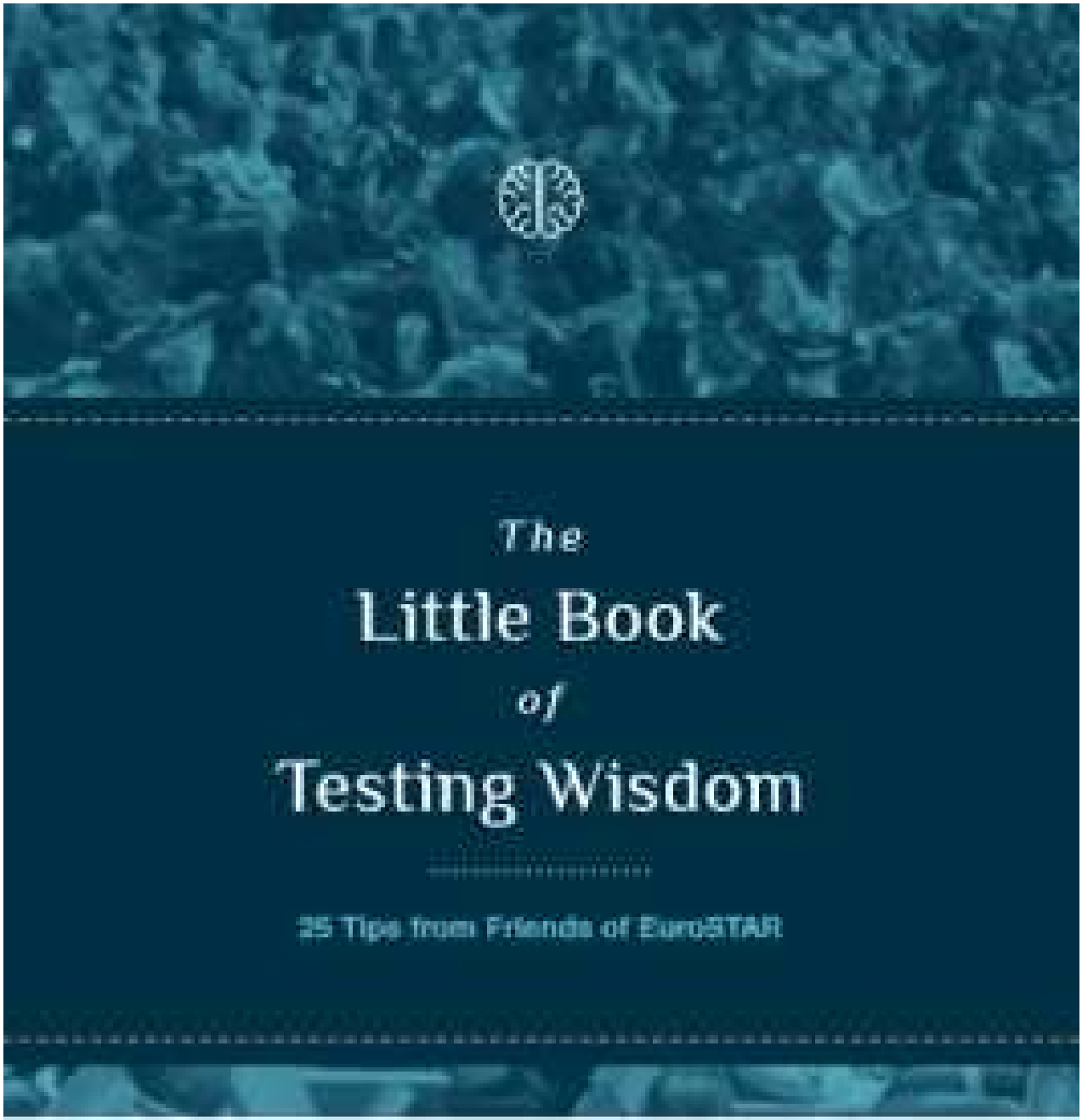


EuroSTAR
Software Testing Conference

Practice "real" Risk-Based Testing, *but keep it simple*

When I was asked the question "What is the top tip or piece of advice that has been most constant in your career that you would share with fellow testers?", it automatically made me think and look back to many projects I was involved in. Already back in 1984 (!!) Bill Hetzel stated "Testing is Risk-Based" [1] and a popular statement by Martin Pol was always "No Risk = No Test". The amount of testing performed depends on the risks involved. Risk must be used as the basis for allocating the test effort that is available and for selecting what to test and where to place emphasis. One would think that after all these years the testing discipline has mastered risk-based testing. I have seen some great implementations and best-practices, but unfortunately still today I encounter many projects that struggle with risk-based testing.

Introduction

It can be defended easily that software engineering is still in its childhood. Although less projects end in total failure than in the early days of IT, users, customers, and the business are still frequently not completely satisfied, if not disappointed, with the end result. This problem touches the essence of risk management, as risk management in general focuses on ways to reduce the discrepancies between the intended and the actual outcome. For the testing discipline, risk-based testing is the practical answer to the required implementation of product risk management in ICT projects.

Testing often has to be done under severe time pressure. This may be due to delays in the activities prior to testing, scope expansion, resource restrictions and many other causes. The answer to all of this is a differentiated risk-based test approach in order to do the best possible job within constraints. Which part of the system requires the most attention? There is not one answer, and decisions about what to test have to be risk-based.

Most projects apply some kind of (implicit) risk-based testing. We all have to balance between product quality, spending testing effort and deadlines. Whether in a traditional waterfall or V-model environment, or applying Agile development methodologies, e.g., SCRUM, establishing clear and agreed testing priorities is always a challenge. Risk-based testing is the basis of almost every testing activity. Of course risk-based testing and setting priorities should be driven by business objectives. Testing nor the Agile team are the risk owner, but the products' stakeholders are. It is our or the teams' job to inform the stakeholders about risk-based decisions and provide visibility on product risk status. Risk-based testing starts by doing risk identification and analysis in close co-operation with stakeholders, e.g., the product owner. It also addresses the mitigation approach regarding the identified product risks.

Unfortunately still today after all these years, many projects are struggling to do proper risk-based testing that allows them to be effective at finding defects and not waste effort on things that are less important. I encounter projects that do risk-based testing by the book, following some process whereby the process seems to be more important than the result. Often it is made too difficult, too big to fit (especially in today's Agile context), not really used as a basis for defining the test strategy and test approach, and/or a risk analysis is done but everything is testing in the same way. In short, I observe many project that do risk-based testing "in name only" without real added value. I run courses and do consultancy on this topic all over the world and I'm surprised every time again regarding the lack of knowledge and practical skills.

Lessons Learned

Of course this is a paper only, and not a full book and thus from many practical experiences in various domains, I would like to share 10 essential lessons learned regarding risk-based testing; 10 things to remember and get you started.

1. **Start risk-analysis by doing a proper stakeholder analysis**. Since stakeholders provide the essential information for the identification and analysis of risks, having the right set of stakeholders is essential. In Utopia a thorough stakeholder analysis has already taken place during requirements phase. Both stakeholders from a business perspective and from a technical perspective (e.g., architect, lead engineer) are required. Although the product owner is typically a good place to start, there may be other people in the organization that need to contribute from a business perspective. Remember, a forgotten stakeholder implies forgotten risks.

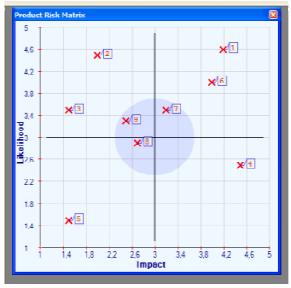
2. **State the product risks in a business language**. Communication is vital to a successful project. Product risks should be stated in such a way that they are understood by the business. It should be clear to them what it means if a risk becomes apparent. Only product risks where all understand what the impact is, in case of a failure, will get focus in communication. It recommend to start the risk process from requirements (user stories), these should already be stated in a user/business language.

3. **Recognize that impact and likelihood are different**. Some product risks analysis techniques calculate the level of risk by multiplying impact by likelihood and from then on just using the resulting calculated risk level. This is clearly wrong and dangerous! An extremely high impact risk (e.g. safety) with a low likelihood may then not get any or too little attention. Impact usually relates to business factors and business risks, likelihood relates to technical factors and technical risks. Consider the following table where impact and likelihood are both rated on a scale from 1 to 10. Although the

multiplication shows the same result for risk A as for risk B, they are clearly very different by nature very different and should also have different mitigation approach.

	Impact	Likelihood	Risk Level
Product Risk A	1	10	10
Product Risk B	10	1	10

4. **Visualize the results of the product risk analysis**. A picture is worth more than a thousand words. Presenting the results in a diagram is usually much more clear to stakeholders than a table (excel sheet) with many numbers. The latter becomes unreadable very easily and some loose themselves in a number based discussion. A recommendation, and often used, is the so-called product risk matrix. The product risk matrix has impact on the horizontal axis, likelihood on the vertical axis, and four quadrants each represent a level and type of risk. A picture, e.g., the product risk matrix, generally provides a much better basis for discussing and validating the product risks.



PRISMA Risk Matrix

5. **Consider both functional and non-functional risks**. Much like with requirements some tend to "forget" the non-functional product risks. However, in practice the non-functional quality attributes, such as performance, reliability and usability, most often make a huge difference. Beware not to go overboard and lose yourself in long and detailed non-functional list of quality attributes, e.g., co-existence, replaceability, changeability, etc., that only few really understand. Only discuss a limited set of non-functional quality attributes, preferably in a business language, that could be important to the product under test, and that you are capable of testing.

6. **Define a differentiated risk-based test approach**. Product risks that are more critical than others should be tested differently, e.g., with more coverage, stricter exit criteria etc. A tester or developer testing an item related to critical product risk should test this differently than testing a less critical item. This differentiated risk-based test approach should be clearly defined upfront to allow for effective and efficient usage of test resources. Unfortunately many projects that do a product risk analysis do not define the implementation of it - a differentiated risk-based test approach. They leave it up to those who are testing what it means if something is more critical and how it should be tested. In practice most testers and developers do not have sufficient knowledge and

skills for this and need some guidance. Examples of and how to define a differentiated test approach can be found in [2].

7. **Track progress against the identified product risks**. Many do a product risk analysis but progress tracking and reports are again defect based. Stakeholders, e.g., the product owner, tell us which product risks are important and should be mitigated before being able to release the system. A test status report, either in writing or on a task board, should provide this information and support the release decision. In practice, defect based reports are often not the most usable for business stakeholders. It is recommended to define product risk coverage and product risks mitigated as part of the exit or definition-of-done criteria.

8. **Choose the product risk analysis method that meets your needs**. Many methods on product risk analysis are not light weight and extremely thorough, unfortunately I also find many (if not most) methods to be highly theoretical and not very practical. Some thorough methods, e.g., FMEA [3], may fit when doing testing in a traditional environment for a safety critical system. On the other side of the spectrum, when doing testing in Agile context it is still important to make choices since we still cannot test everything. However, the product-risk session should be light weight and very focused. A simple brainstorm may suffice at the beginning of an iteration. Risk-Poker [4] is another popular risk method for Agile projects. In general, don't make it more difficult than necessary.

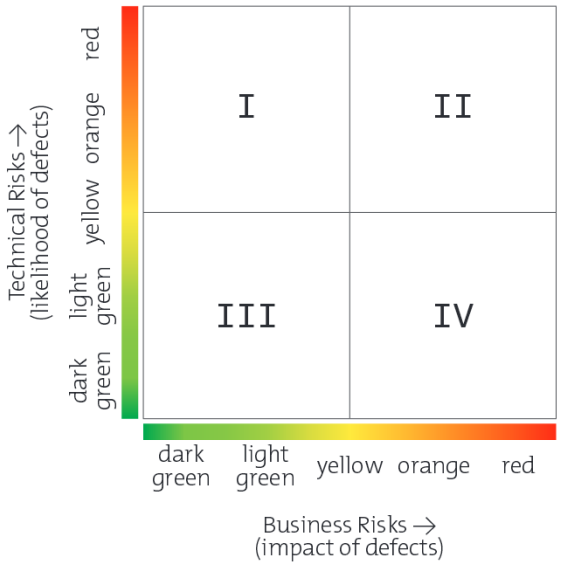
9. **Revisit the product risks on a regular basis**. The product risk identification and analysis are based on stakeholders' perceptions and expectations. These will change over time. Early testing will reveal some new risks while mitigating others. Exploratory testing will typically identify new risks in the product. Changing requirements (user stories) often means changed product risks. The product risk list is by no means stable throughout a project or an iteration. It pays off to revisit the product risk list regularly, e.g., in traditional projects at least at every project milestone.

10. **Establish clear risk ownership and responsibilities**. In many projects testers identify and analyze the risks. This is wrong; testers are not the risk owners!! Our responsibility is 'only' to facilitate the risk analysis process and inform our stakeholders on the status of the product risks. In Agile this becomes the responsibility of the team, whereby the team is responsible for the quality of the product. When stakeholders, possibly represented by the product owner, are asked to identify product risks and thereby indicate what to test and what no to test, they suddenly become aware that they are

the deciding factor. If they do it wrong (e.g., forget a product risk), they are to blame and not the tester or Agile team. This often leads to stakeholders' resistance: „It was so easy when the tester took the decision for us."

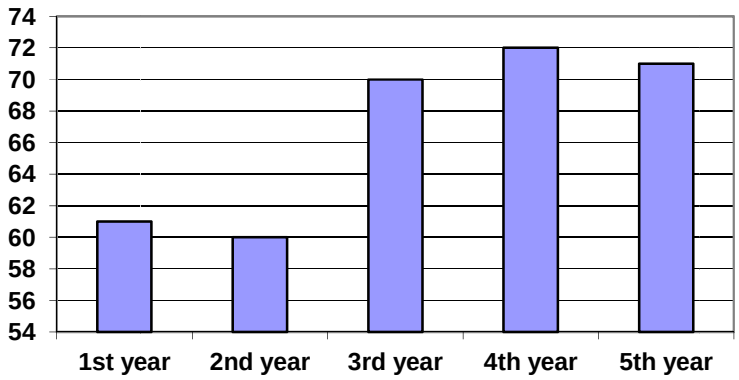
Agile: One page test plan

Some Agile people consider risk-based testing "old school". Hopefully it is already clear by the reader that this totally wrong, we "just" need to tune the approach to fit the context. In many Agile projects the product risk matrix including a defined differentiated approach are used as the "test plan" for the next iteration. By putting a picture on the wall, everyone can see the test actions to be performed. This picture is often enhanced by providing the Definition of Done criteria per quadrant. A short product risk session during iteration planning delivers the iteration "test plan" in an easily readable format on one page. How much more efficient and effective can one become!?



Results on Product Quality

I like numbers and facts, therefore a case study reference to add to this paper. An organization I was involved in for many years introduced risk-based testing as one of their core methods for product risk management, tracked defect numbers and calculated Defect Detection Percentage (DDP) for several years at system test level. Defect Detection Percentage was defined as "the number of defects found by a test phase, divided by the number found by that test phase and any other means afterwards". As one can see from figure below their DDP improved by over 10% after the introduction of risk-based testing in year 2. Interesting enough at the same time they were even able to also reduce the test execution effort and lead time due to more focused testing.



Defect Detection Percentage (DDP) System Testing

References

[1] B. Hetzel (1984), *The complete guide to software testing 2nd edition,* QED Information Sciences, Inc.

[2] E. van Veenendaal (2012), *Practical Risk-Based Testing: The PRISMA Approach*, UTN Publishers

[3] R. McDermott, R.J. Mikulak and M.R. Beaugard (1996), *The basics of FMEA,* Productivity Press

[4] E. van Veenendaal (2016), *Practical Risk-Based Testig with PRISMA (also for Agile projects)*, in: Quality Matters, August 2016

About the author

Erik van Veenendaal (www.erikvanveenendaal.nl) is leading international consultant, trainer, and recognized expert in software testing and quality. Erik is the author of a number of books and papers, one of the core developers of the TMap testing methodology and the TMMi improvement model, a participant in working parties of the International Software Testing Qualification Board (ISTQB), and currently CEO of the TMMi Foundation. He is a frequent keynote and tutorial speaker at international testing conferences. For his major contributions to the field of testing, Erik received the European Testing Excellence Award (2007) and the ISTQB International Software Testing Excellence Award (2015). You can follow Erik via twitter @ErikvVeenendaal.